Kinetic Monte Carlo Modeling of Graphene Sheet Growth During CVD

Hironori Kondo[†]

[†]Harvard College, Harvard University
1954 Harvard Yard Mail Center, 1 Oxford Street, Cambridge, MA 02138
Author email: *hironorikondo92@gmail.com*

Abstract

Graphene is a material of key interest across several research fields. Bulk graphene synthesis, however, has long remained a challenge for larger-scale projects and real-world manufacturability. This work seeks an improved understanding of graphene sheet growth via computational modeling, with the objective of maximizing grain size. To this end, the kinetic Monte Carlo method is used to simulate chemical vapor deposition under various configurations of carbon flow and graphene seeding. Ultimately, both quantitative and qualitative results are obtained to shed light on graphene growth mechanisms, with insights into real-world synthesis and future computational models.

Introduction and Background

In recent years, graphene has attracted much attention within the research community. In addition to its mechanical advantages, the material has demonstrated promising electrical properties when used in photovoltaics,[1] hydrovoltaics,[2] transistors,[1,3] lithium-ion batteries,[4] and pressure-retarded osmosis systems,[5] among other applications. Bulk synthesis of graphene hence becomes of high interest, both to realize larger-scale research projects and to enable real-world manufacturing.

One popular method of graphene synthesis is via chemical vapor deposition (CVD; see Appendix A for details) on copper (Cu) subtrates,[2,3,6,7] which have been found to yield more uniform monolayers than other substrates.[1] The challenge, however, is in producing larger *grain sizes* via CVD,[1,7] as this dictates the amount of usable, uniform graphene produced. Hence, a better understanding of graphene sheet growth is highly desired to expand the material's practical impact.[1,8]

To this end, this work computationally models graphene sheet growth during CVD on a Cu substrate, analyzing the effects of "pre-seeding" (Fig. 1) and carbon flow rate on grain size. At its most general, the model focuses on four forms of movement (henceforth "events") within a pre-defined hexagonal lattice: incoming carbon flow, surface diffusion, carbon attachment, and carbon detachment (Fig. 2). To simulate these events, we follow a four-step process:

1) randomly seed carbon "atoms", 2) count the number of neighbors of each atom, 3) apply the *kinetic Monte Carlo* (kMC)[8,9] method, and 4) repeat the cycle across several iterations.

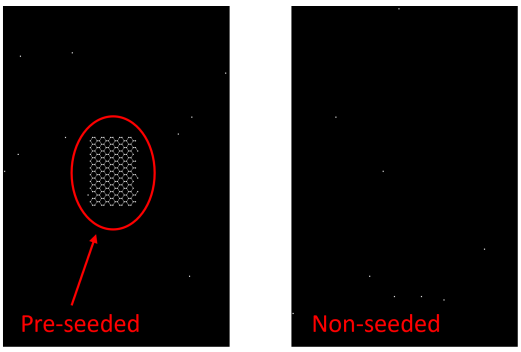

Figure 1. Comparison of pre-seeded (left) and non-seeded (right) samples.

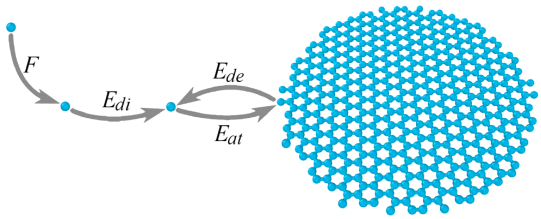

Figure 2. Schematic of all kMC events. F *represents the flow rate of carbon atoms into the CVD chamber,* E_{di} *diffusion along the substrate surface,* E_{at} *bond formation (i.e., attachment), and* E_{de} *bond breakage (i.e., detachment). Adapted from Chen* et al. *2020.*[8]

We begin by randomly seeding carbon "atoms" into a simulated CVD chamber, modeled as a two-dimensional "box". This mimics the incoming flow of carbon, where the number of seeded "atoms" at each iteration corresponds to the physical flow rate. From there, we count the number of neighbors of each atom within the box, which reveals the number of possible events (*e.g.*, surface diffusion, forming one bond, forming two bonds, *etc.*). From there, we can apply kMC, which relates the *energy* of each event to a corresponding probability determined by the Boltzmann distribution. Based on these probabilities, one particular event is chosen for each atom, completing one "iteration" of the simulation (details of this calculation are expanded upon in Appendix B). The entire process is then repeated until the desired number of carbon atoms has been introduced. Hence, we are able to simulate the *movement* of individual atoms based on the energy required for each event, which affects the probability of said event occurring.

For our Cu substrate specifically, these necessary energies are well-documented in the existing literature. In particular, we use values derived from first principles by Wu *et al.* 2015.[10] Critically, we make four major approximations: 1) all edge diffusion is disregarded; 2) all dimer movement is disregarded, making single-atom events the only source of movement; 3) the event energies are proportional to the *number* of bonds being formed or broken, independent of the domain geometry; and 4) the graphene is strictly a monolayer, with the simulation restricted to two dimensions and no "stacking" of atoms permitted. Approximations one through three are visualized below in Figures 3 to 5, which are adapted from Chen *et al.* 2020.[8]

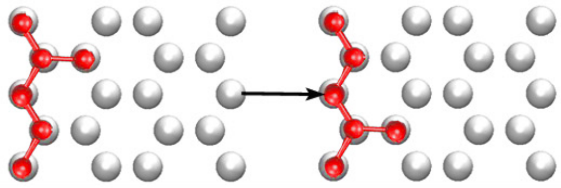

Figure 3. A visualization of edge diffusion, wherein the carbon atom moves along the edge of a graphene domain in a single event. We instead approximate this by breaking the bond and forming a new one in two separate *events.*

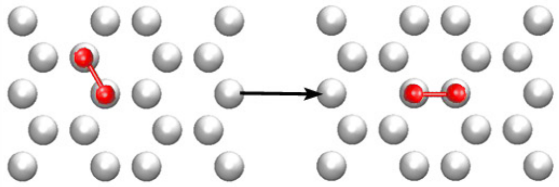

Figure 4. A visualization of dimer rotation. This is not possible within our model; for such a movement to occur, the dimer must break apart, and the atoms must move independently to form a new bond in the new position.

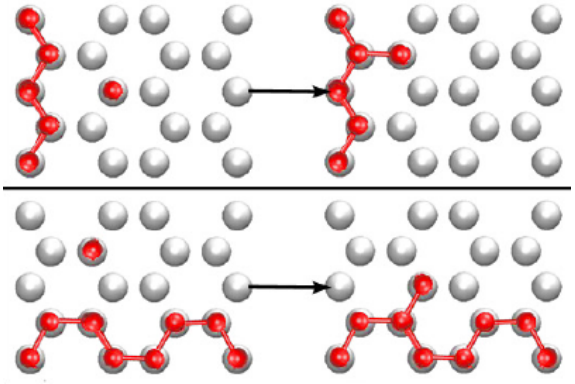

Figure 5. A visualization of carbon atoms binding to different domain geometries. From first principles, these two events have different energies, due to the differences in the surfaces the atoms are attaching to. We approximate these two events to be the same, based on the fact that only one bond is forming.

To conduct our analysis, we implemented the above procedure in MATLAB R2021a with the following parameters:

- **Chamber size:** 100 by 100 pixels
- **Chamber temperature:** 1000 °C (derived from existing literature[1,8])
- **Possible atom locations:** 5000 in total (based on the pre-defined hexagonal lattice)
- **Initial configuration:** Pre-seeded, non-seeded (conducted identically besides the presence or omission of a pre-seed)
- **Flow rate:** 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9 atoms per iteration (10 total variations of flow rate)
- **Total number of atoms introduced:** 2048 (*sc.* the number of iterations for each flow rate was different, and each "run", or "cycle", of the simulation ended when 2048 atoms were reached)
- **Number of cycles per flow & seeding variation:** 500 (*i.e.*, each unique configuration was run 500 separate times; each run was comprised of several iterations and ended when 2048 atoms were introduced)

These parameters made for 500 independent runs × 10 different flow rates × two different initial configurations, totalling 10,000 independent trials (see Appendix C for details on the MATLAB implementation). With this information, we can determine how flow rates affect graphene grain size for both pre-seeded and non-seeded cases. Quantitatively, this was done by taking the arithmetic mean of each result across all 500 of its runs, then plotting the changes in grain sizes and grain counts (*i.e.*, the number of individual graphene grains). For qualitative analysis, we converted the runs into images of graphene lattices for visualization. Finally, we abstracted the findings from both analyses to the overall grain size distributions, providing a better big-picture view of the simulation results.

Results and Discussion

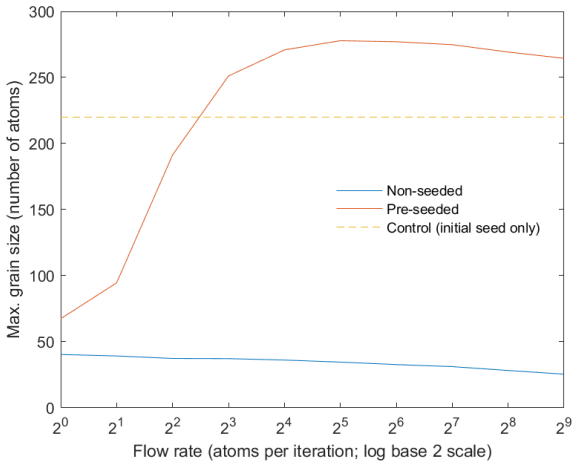

Figure 6. Mean maximum grain size by carbon flow rate, with a control (the atom count of the initial seed).

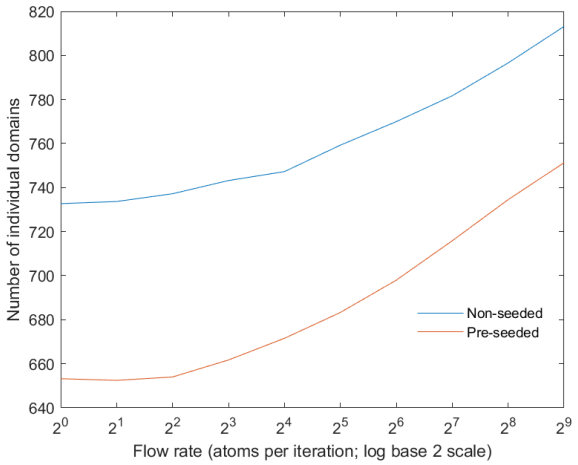

Figure 7. Mean grain count (number of individual grains) by carbon flow rate.

We begin with our quantitative results. Figures 6 and 7 show how mean maximum grain size and mean grain count vary by flow rate, respectively (reprinted larger in Appendix D). These results shed light into the mechanisms at play during graphene sheet growth.

For Fig. 6, we observe that the pre-seeded samples have consistently higher grain sizes than the non-seeded samples. In particular, the trends in the two clearly diverge: the pre-seeded samples are shown to increase and then decrease in grain size, while the non-seeded ones strictly decrease. Hence, in seeking larger graphene sheets, seeding is essential to 1) producing larger grains and 2) enabling significant increases in growth by controlling the flow rate.

We observe, however, that the pre-seeded samples do not always result in graphene *growth*. In particular, our first measured instance of growth was at a flow rate of 2^3 atoms/iteration; every sample below this flow rate resulted in sheet *shrinkage*. A potential explanation for this phenomenon is that, due to the higher number of iterations necessitated by the lower flow rates, the atoms in the initial sheet are given more opportunities to diffuse away, leading to an overall decrease in grain size.

We also note that, for the pre-seeded samples, grain size does not *strictly increase* with flow rate. Instead, there is a local maximum at 2^5 atoms/iteration, after which the grain size gradually tapers off. This phenomenon may again be explained by diffusion; instead of the sheet fragmenting however, the now *lower* number of iterations are not granting newly-seeded atoms enough opportunity to diffuse *towards* the sheet and bind to it. In essence, the graphene sheet becomes surrounded by loose carbon atoms that have not diffused and gathered into concrete grains.

This finding is supported by Fig. 7, wherein the number of grains increases with flow rate for both the pre-seeded and non-seeded cases. Hence, our notion that the number of loose atoms *increases* with flow rate is supported, as the total number of seeded atoms is consistent (*i.e.*,

2048), but the number of grains is increasing. As a note, we can also observe that the pre-seeded configurations have much fewer grains than the non-seeded ones, which aligns with what Fig. 6 has indicated about maximum grain sizes (*sc.* for the pre-seeded samples, there are more atoms within a *single* grain and hence fewer "loose" atoms).

All of these observations and explanations are reflected in our qualitative analysis, summarized below in Figures 8 to 13:

Figure 8. Non-seeded run with a flow rate of 8 atoms/iteration after one iteration.

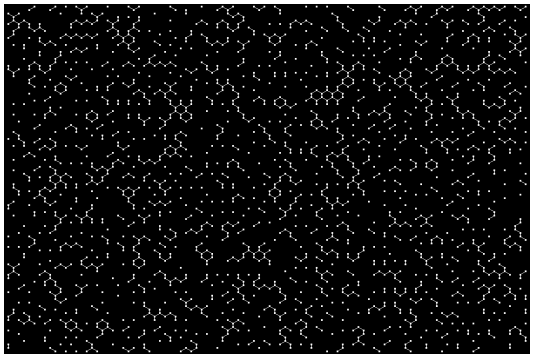

Figure 9. Non-seeded run with a flow rate of 8 atoms/iteration after 256 iterations.

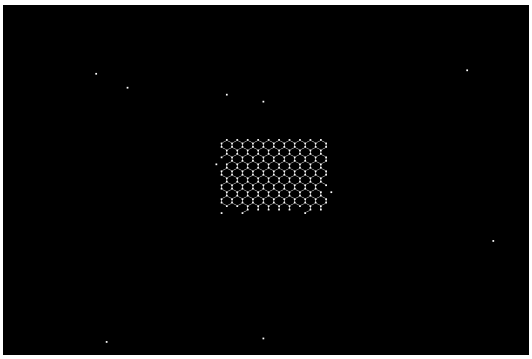

Figure 10. Pre-seeded run with a flow rate of 8 atoms/iteration after one iteration.

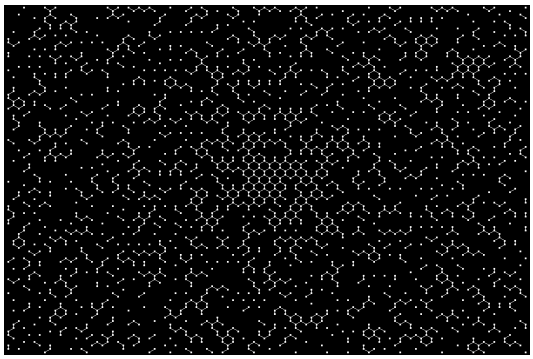

Figure 11. Pre-seeded run with a flow rate of 8 atoms/iteration after 256 iterations.

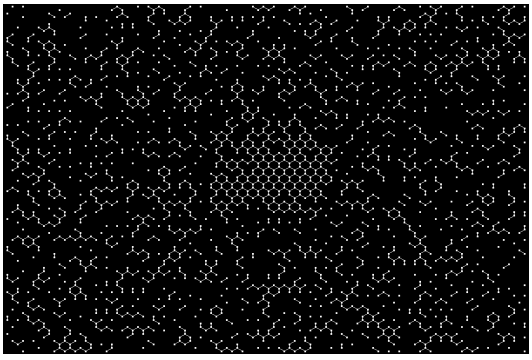

Figure 12. Pre-seeded run with a flow rate of 32 atoms/iteration after 64 iterations.

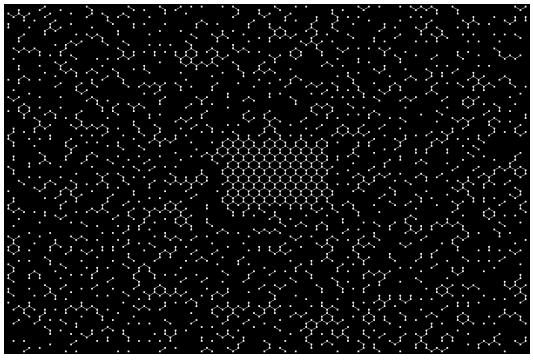

Figure 13. Pre-seeded run with a flow rate of 512 atoms/iteration after 4 iterations.

These visualizations support the two overarching notions from our quantitative analysis:

1. Pre-seeding is instrumental to developing larger graphene grains, as the lack of a pre-seed creates smaller grains (Fig. 6) and scattered carbon atoms (Fig. 7). For instance, we see how a pre-seeded sample yields a denser congregation of atoms towards the middle (Fig. 11), whereas a non-seeded sample becomes sparse and fragmented (Fig. 9), despite the same flow conditions in both cases.

2. Diffusion (which is related to flow rate) controls the interplay between sheet fragmentation and growth (Fig. 6). We observe that, at a low flow rate, an initially rectangular pre-seed (Fig. 10) becomes more sparse (Fig. 11), indicating an outward diffusion of atoms. If we increase the flow rate, this outward diffusion is reduced and the sheet ends up larger (Fig. 12); increase the flow rate *too much*, however, and the product ends up looking remarkably similar to the pre-seed (Fig. 13). In essence, the atoms are not given opportunities to flow towards the largest grain and cause it to grow. Hence, a delicate balance must be struck, wherein atoms are given opportunities to bind to the grain but not diffuse outward by too much.

Hence, our observations are consistent between both the quantitative and the qualitative results. To further extend these findings, we can abstract them to their effects on the overall *distribution* of grain sizes (reprinted larger in Appendix D):

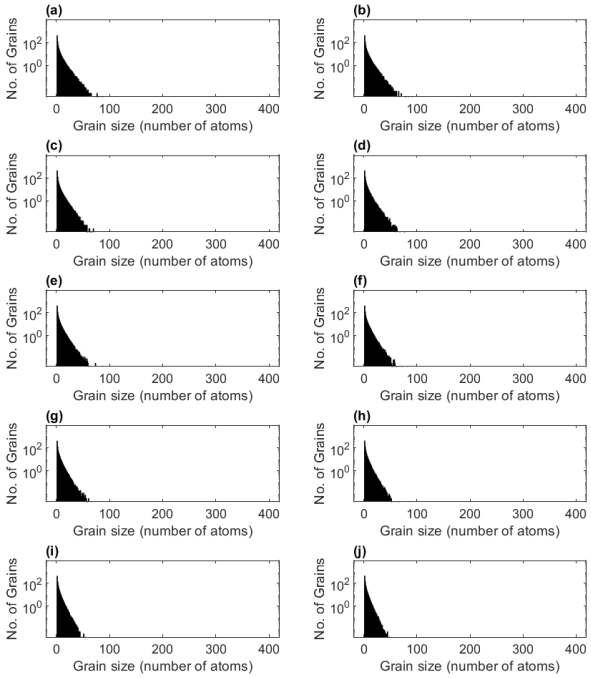

Figure 14. Histogram of mean non-seeded grain size distribution, for flow rates of (a) 2^0, (b) 2^1, (c) 2^2, (d) 2^3, (e) 2^4, (f) 2^5, (g) 2^6, (h) 2^7, (i) 2^8, (j) 2^9 *atoms per iteration.*

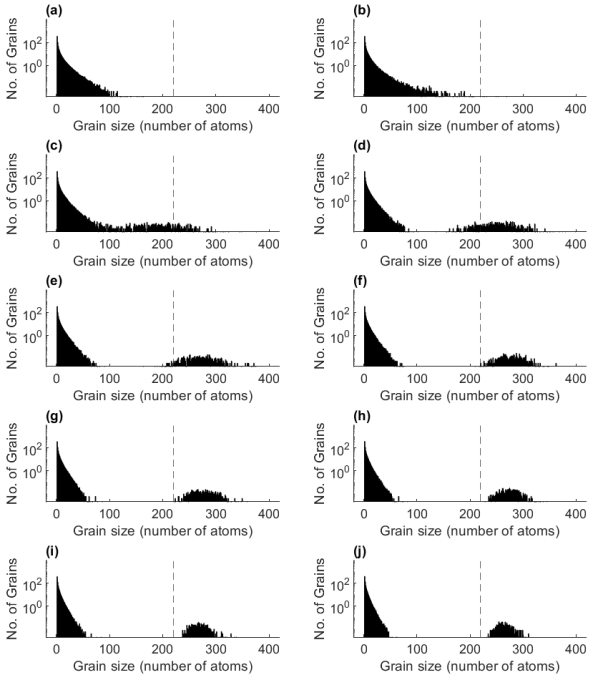

Figure 15. Histogram of mean pre-seeded grain size distribution. Dashed lines indicate initial pre-seed size. Refer to Fig. 14 for the respective flow rates of the subplots.

Here, we have two key observations, which are taken in parallel to our previous two points:

1. The pre-seeded samples form a distinct "bump" in their distributions at higher grain sizes (Fig. 15). This is a reflection of our previous notion that pre-seeding is essential to creating larger-grain graphene sheets.

2. Higher flow rates cause horizontal compressions in the distributions, making for sharper, cleaner peaks (Fig. 14, 15). For the pre-seeded samples, lower flow rates do not yield distinguishable/sharp "bumps" in the higher-grain-size region (Fig. 15). These both reveal the role of diffusion, as previously described:

   The horizontal compressions suggest that higher flow rates create less room for variability in the grain size distribution. This in turn suggests that fewer opportunities (*i.e.,* iterations) are provided for atom movement, hence yielding a more fragmented product characterized by more small-size grains. Previously, we observed the same phenomenon in Fig. 7, 13.

   The formation of the high-grain-size "bump" at high flow rates suggests that, at low flow rates, even the presence of a seed results in smaller final grains. This implies that the initial seeds fragment and disperse during the CVD process, a symptom of too much atom movement (*i.e.,* too many iterations that permit too much outward diffusion). We previously observed the same phenomenon in Fig. 6, 11.

Hence, we observe that our quantitative and qualitative conclusions are reflected within the overall distributions of grain sizes. We thus have three distinct perspectives from which the same major findings can be derived, strengthening the overall case for our core notions.

Conclusion and Outlook

We have thus determined that pre-seeding is instrumental to grain growth, and flow rate controls the interplay between fragmentation (*i.e.,* outward diffusion) and binding (*i.e.,* inward diffusion and bond formation). Critically, we saw that lower flow rates create more opportunity for diffusion, leading to sparser products; higher flow rates, meanwhile, limit the amount of change from the initial seed, hence making for a less productive process. A balance between the two must be struck to maximize grain size, as depicted in Fig. 6. These findings are supported by both quantitative and qualitative analyses, and they are reflected in the overall grain size distributions of our samples.

The approximations of our model lead to limitations that should be addressed in future works. In particular, all samples show a high number of individual grains (Fig. 7), most of which are very small (Fig. 14, 15). As this trait is not entirely consistent with the existing literature,[8] the omissions of dimer movement and edge diffusion may be limiting the ability of the carbon atoms to congregate and form larger, uniform sheets. Future models should attempt to incorporate these kMC events, but implementation will be more involved.

In addition, while this work addressed CVD on Cu films specifically, its core structure can be adapted to other graphene synthesis methods. These include CVD

on other metal films, non-monolayer processes, and other technologies such as Molecular Beam Epitaxy. Hence, in short, the conclusions herein yield insight into not only the mechanisms of graphene sheet growth, but also expandability to other synthesis processes and more thorough models of atom movement.

Acknowledgements

The author thanks Alvin Hsu, Liu Group, Harvard University Department of Chemistry and Chemical Biology (CCB) for advising this work. The author also thanks Senior Preceptor Lu Wang, Professor Kang-Kuen Ni of Harvard CCB and members of the PS 10 teaching staff for their guidance and supervision.

References

(1) Zhang, Y.; Zhang, L.; Zhou, C. Review of Chemical Vapor Deposition of Graphene and Related Applications. *Acc. Chem. Res.* **2013**, *46*, 2329–2339, DOI: 10.1021/ar300203n.

(2) Zhang, Z.; Li, X.; Yin, J.; Xu, Y.; Fei, W.; Xue, M.; Wang, Q.; Zhou, J.; Guo, W. Emerging hydrovoltaic technology. *Nature Nanotech* **2018**, *13*, 1109–1119, DOI: 10.1038/s41565-018-0228-6.

(3) Li, X.; Cai, W.; An, J.; Kim, S.; Nah, J.; Yang, D.; Piner, R.; Velamakanni, A.; Jung, I.; Tutuc, E.; Banerjee, S. K.; Colombo, L.; Ruoff, R. S. Large-Area Synthesis of High-Quality and Uniform Graphene Films on Copper Foils. *Science* **2009**, *324*, 1312–1314, DOI: 10.1126/science.1171245.

(4) Kucinskis, G.; Bajars, G.; Kleperis, J. Graphene in lithium ion battery cathode materials: A review. *Journal of Power Sources* **2013**, *240*, 66–79, DOI: 10.1016/j.jpowsour.2013.03.160.

(5) Jia, Z.; Wang, B.; Song, S.; Fan, Y. Blue energy: Current technologies for sustainable power generation from water salinity gradient. *Renewable and Sustainable Energy Reviews* **2014**, *31*, 91–100, DOI: 10.1016/j.rser.2013.11.049.

(6) Xu, S.; Zhang, L.; Wang, B.; Ruoff, R. S. Chemical vapor deposition of graphene on thin-metal films. *Cell Reports Physical Science* **2021**, *2*, 100372, DOI: 10.1016/j.xcrp.2021.100372.

(7) Gao, L.; Guest, J. R.; Guisinger, N. P. Epitaxial Graphene on Cu(111). *Nano Lett.* **2010**, *10*, 3512–3516, DOI: 10.1021/nl1016706.

(8) Chen, S.; Gao, J.; Srinivasan, B. M.; Zhang, G.; Sorkin, V.; Hariharaputran, R.; Zhang, Y.-W. An all-atom kinetic Monte Carlo model for chemical vapor deposition growth of graphene on Cu(1 1 1) substrate. **2020**, *32*, 155401, DOI: 10.1088/1361-648X/ab62bf.

(9) Voter, A. F. In *Radiation Effects in Solids*, ed. by Sickafus, K. E.; Kotomin, E. A.; Uberuaga, B. P., Springer, Dordrecht: 2007, pp 1–23, DOI: 10.1007/978-1-4020-5295-8_1.

(10) Wu, P.; Zhang, Y.; Cui, P.; Li, Z.; Yang, J.; Zhang, Z. Carbon Dimers as the Dominant Feeding Species in Epitaxial Growth and Morphological Phase Transition of Graphene on Different Cu Substrates. *Phys. Rev. Lett.* **2015**, *114*, 216102, DOI: 10.1103/PhysRevLett.114.216102.

A CVD Fundamentals

This work assumes a basic understanding of the CVD process; the purpose of this section is to fill in the fundamentals necessary to understand the computation model. For a more detailed look at CVD synthesis of graphene on metal film substrates, please refer to Zhang *et al.* 2013.[1]

At its most basic, CVD graphene synthesis is comprised of three elements:

1. A gaseous carbon source (*e.g.,* CH_4)
2. A heated chamber (typically tubular)
3. A substrate

The carbon source is introduced into the chamber at a certain rate, described in our model as the "flow rate". In the heat of the chamber, the gas decomposes to yield pure carbon, which deposits onto the substrate. A graphene sheet can thus begin to form.

The purpose of the substrate is to provide a surface onto which the carbon can attach. It also behaves as a catalyst to lower the energy barrier of graphene formation, along with dictating the deposition mechanism.[1] Within our model, this manifested as the discretization of the chamber (*sc.* by pre-defining a hexagonal lattice), as well as the specific event energies used within the kMC portion.

Though elementary, this information is sufficient to understand our model. It is by no means exhaustive in terms of graphene synthesis, and it certainly does not cover all forms and applications of CVD. Instead, it merely serves to contextualize the methodology of this work.

B Kinetic Monte Carlo

Previously, this work outlined the core tenets of the kinetic Monte Carlo method. The purpose of this section is to elaborate in further detail the specific mechanisms of kMC[8,9] as it relates to our model.

The basis of kMC lies in *energy.* Carbon atoms interacting with the Cu substrate and with each other require energy; different interactions, or "events", require different amounts of energy, which can be derived from first principles.[10]

kMC begins by relating each event i to a given rate k_i, related to the event's energy E_i via the *Boltzmann distribution*

$$k_i = \nu \exp(-\frac{E_i}{k_b T}),$$

wherein ν is the frequency of atomic vibration (here 1.0×10^{12} s^{-1} as per existing literature[8]), k_b is the Boltzmann constant, and T is the chamber temperature (here chosen to be 1000 °C, again as per existing literature[1,8]). The sum of all possible rates for a given atom at a given iteration is denoted by k_{tot}.

These rates are used to weight a random number generator, such that the generator selects events with probabilities corresponding to the events' rates. To do so, we take the rates $k_1, \ldots, k_M$ of all possible events $1, \ldots, M$ for a given atom and stack them end-to-end in an array, of which we can then take the cumulative sum (*sc.* the first entry is k_1, the second is $k_1 + k_2$, and the h^{th} is $k_1 + \ldots + k_h$). By drawing a random number r, we can select the first event h at which the partial sum is less than or equal to rk_{tot} (*i.e.,* h is the greatest element of $\{1, \ldots, M\}$ for which $(k_1 + \ldots + k_h) \leq rk_{tot}$). This event h is chosen to occur, and the algorithm selects events with the correct probabilities as determined by the Boltzmann distribution.

This algorithm is visualized below in Fig. 16. Note that, in our model, we determined the possible events $1, \ldots, M$ of each atom at each iteration by counting its neighboring atoms, along with each of their respective neighbors (see Appendix C).

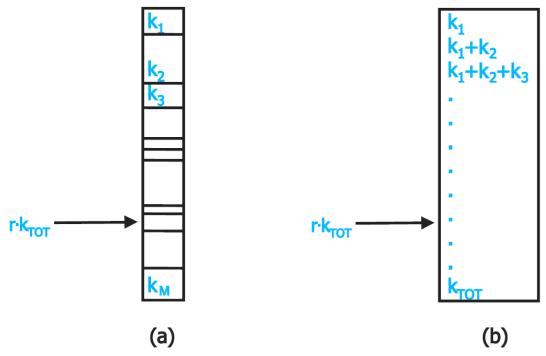

Figure 16. Visualization of the kMC process. (a) *Possible events are assigned rates corresponding to their probabilities, here depicted as length. A random number* r *is drawn, and* $\mathrm{k_{tot}}$ *is the total sum of the rates (*i.e., $\mathrm{k_1 + \ldots + k_M}$*).* (b) *Comparing* $\mathrm{rk_{tot}}$ *to the cumulative sums "weights" the random number generator to select events with the correct probabilities. Adapted from Voter 2007.*[9]

Now, we have covered the fundamentals of how kMC selects events. There is a secondary portion that assigns time lengths to each event, again related to a weighted random number generator. For the purposes of this work, however, comparing across atom counts instead of time yielded more concrete insights, as doing so provides more standardization with which to understand grain sizes. Hence, this time-selection algorithm has been omitted, but there is some relevance to our model, as the total simulation time is related to the *number* of iterations as discussed herein. For details about this time algorithm, refer to Voter 2007[9] and Chen *et al.* 2020.[8]

C MATLAB Code

This section contains the MATLAB R2021a code that runs the simulations. It is a brief, reference-only outline of the model and by no means a complete exposition to its workings.

The code consists of two main elements: a live script that processes all 10,000 runs and gathers information about them, along with a function stored in a standard script that handles each individual run (which consists of several iterations). The rough breakdown is as follows:

- Live script
    1. Initialize the variables
    2. Track the indices of adjacent lattice points (*i.e.,* neighboring positions)
    3. Define a vector to map the lattice to a single-column vector
    4. Run 500 non-seeded cycles of the simulation (by calling the function)
    5. Gather pixel clusters per cycle
    6. Sort the information into arrays
    7. Repeat steps 4-6 for pre-seeded samples
    8. Gather information from the initial pre-seed (as a control)
    9. Export the workspace variables
- Function
    1. Gather the input variables & define the physical constants
    2. Randomly seed atoms
    3. Count neighbors for each atom
    4. Run the kMC algorithm
    5. Output the final CVD "box"

We begin by showing the live script, followed directly by the function:

```
clear all

%generate CVD substrate
S = zeros(102,102);

%generate hexagonal lattice
S(2:101,1:100) = repmat([1 1 0 0;0 0 1 1], 50, 25);
S(:,101)=S(:,1);
S(:,1) = 0;

%create array with indices of lattice pts; the map_n is the location in S
map = find(S==1);

%inversemap_n, where n is a location in S, is a location in G
inversemap = zeros(max(map),1);
for ind = 1:5000
    inversemap(map(ind)) = ind;
end

%create contact lattice (tracking indices of adjacent lattice pts)
C = zeros(5000,1,3);
i = 0;
for indclm = 2:101
    for indrow = 2:101
        if S(indrow,indclm)==1
            i = i+1;
            SSub=zeros(102,102);
            SSub((indrow-1):(indrow+1),(indclm-1):(indclm+1)) = S((indrow-1):(indrow+1),(indclm-1):(indclm+1));
            SSub(indrow,indclm)=0;
            loc = inversemap(SSub==1);
            locraw = padarray(loc,3-length(loc),0, 'post');
            C(i,1,:)=permute(locraw,[3 2 1]);
        end
    end
end

%create blank graphene sheet for non-seeded samples
G = zeros(5000,1);
%create blank graphene sheet for pre-seeded samples
GS = zeros(5000,1);

%generate pre-seed
SSub=zeros(102,102);
SSub(40:60,40:60)=S(40:60,40:60);
GS(inversemap(SSub==1)) = 1;

%define paramters for the simulation
vari = [1 2 4 8 16 32 64 128 256 512;2048 1024 512 256 128 64 32 16 8 4];
numIt = 500;
numVari = length(vari);

%allocate memory for necessary simulation variables
avGmax = zeros(1,numVari);
avGnum = zeros(1,numVari);
histList = zeros(1,numVari);
avGmaxS = zeros(1,numVari);
avGnumS = zeros(1,numVari);
histListS = zeros(1,numVari);
```

Figure 17. Code for setting up the live script workspace, including defining the hexagonal lattice, tracking adjacent lattice points, allocating memory for variables, defining simulation parameters, etc.

```
for int = 1:numVari
    %extract flow and iterations information
    flow = vari(1,int);
    iterations = vari(2,int);

    %create variable to track list of grain sizes for current configuration
    pixelList = [];

    %run 500 cycles of the same configuration
    for int2 = 1:numIt
        %run simulation
        im = runSim(iterations,flow,G,C,map);

        %extract information about connected atoms
        CC = bwconncomp(im);
        numPixels = cellfun(@numel,CC.PixelIdxList);

        %record list of grain sizes
        pixelList = [pixelList,numPixels];

        %record maximum grain size and grain count
        avGmax(int) = avGmax(int) + max(numPixels);
        avGnum(int) = avGnum(int) + CC.NumObjects;
    end

    %define variables
    pixelList = pixelList';
    histLength = size(histList,1);
    pixelLength = length(pixelList);

    %make sure lengths match
    if histLength < pixelLength
        histList = padarray(histList,pixelLength-histLength,0,'post');
    else
        pixelList = padarray(pixelList,histLength-pixelLength,0,'post');
    end

    %store information into histList
    histList(:,int) = pixelList;

    %update progress
    disp("Tested unseeded vari " + int + " of " + numVari)
end

%take averages of grain sizes and counts
avGmax = avGmax/numIt;
avGnum = avGnum/numIt;
```

Figure 18. Code for running 500 non-seeded cycles of the simulation and gathering the relevant information about them for quantitative analysis.

```
for int = 1:numVari
    %extract flow and iterations information
    flow = vari(1,int);
    iterations = vari(2,int);

    %create variable to track list of grain sizes for current configuration
    pixelList = [];

    %run 500 cycles of the same configuration
    for int2 = 1:numIt
        %run simulation
        im = runSim(iterations,flow,GS,C,map);

        %extract information about connected atoms
        CC = bwconncomp(im);
        numPixels = cellfun(@numel,CC.PixelIdxList);

        %record list of grain sizes
        pixelList = [pixelList,numPixels];

        %record maximum grain size and grain count
        avGmaxS(int) = avGmaxS(int) + max(numPixels);
        avGnumS(int) = avGnumS(int) + CC.NumObjects;
    end

    %define variables
    pixelList = pixelList';
    histLength = size(histListS,1);
    pixelLength = length(pixelList);

    %make sure lengths match
    if histLength < pixelLength
        histListS = padarray(histListS,pixelLength-histLength,0,'post');
    else
        pixelList = padarray(pixelList,histLength-pixelLength,0,'post');
    end

    %store information into histList
    histListS(:,int) = pixelList;

    %update progress
    disp("Tested seeded vari " + int + " of " + numVari)
end

%take averages of grain sizes and counts
avGmaxS = avGmaxS/numIt;
avGnumS = avGnumS/numIt;
```

Figure 19. Code for running 500 pre-seeded cycles of the simulation and gathering the relevant information about them for quantitative analysis.

```
%run same process for an empty chamber with only the seed (control)
im = zeros(102,102);
im(map(GS==1)) = 1;
CC = bwconncomp(im);
numPixels = cellfun(@numel,CC.PixelIdxList);
disp(max(numPixels))
disp(CC.NumObjects)
```

Figure 20. Code for extracting relevant information from the initial pre-seed (as a control, used in Fig. 6, 15).

```
%export variables
save("500_run_2Power.mat")
```

Figure 21. Code for exporting all workspace variables, such that they can be stored and used to make figures without re-running the live script.

This concludes the live script. Next is the function that runs each cycle:

```
%define the code for a single run as a function

function im = runSim(it,fl,GIn,CIn,mapIn)
    %gather input materials
    map = mapIn;
    iterations = it;
    flow = fl;
    GS = GIn;
    C = CIn;

    %define physical constants
    v = 1e12; %Hz, atomic vibration
    kb = 8.617e-5; %eV/K
    kbT = kb*1273; %temperature in K derived from literature
    E = [0 0.9 2.3;0.5 0.25 0.5]; %energies of various events

    %create loop for running the iterations
    for indit = 1:iterations

        %determine available seeding locations
        availablemap=find(GS==0);

        %seeding process
        seed = randi(length(availablemap),1,flow); %generate 10 seed locations
        Gloc = availablemap(seed);
        GS(Gloc) = 1; %create a carbon atom at each seed location in the graphene
```

Figure 22. Code for extracting function inputs, defining physical constants, beginning the iterations, and seeding new carbon atoms.

```
%now begin checking neighbors to get energies and rates
indices = find(GS==1)'; %only execute for occupied locations
for indG = indices

    %define neighbor containers
    NState = C(indG,1,:); %keeps track of locations of unoccupied neighbors
    NMaster = repmat(3,5000,1); %keeps track of number of occupied neighbors

    %begin counting neighbors
    for indz = 1:3
        NLoc = C(indG,1,indz);
        if NLoc ~= 0 %execute if neighbor exists
            if GS(NLoc) == 1 %execute if neighbor occupied
                NState(indz) = 0; %remove neighbor index from NState
            else %execute if neighbor unoccupied
                NMaster(indG) = NMaster(indG)-1; %free up one neighbor
            end
        end
    end

    %count neighbors of each neighbor (same process as before)
    imN = C(indG,1,:);
    imN(imN==0) = []; %remove null entries (e.g., at sheet edges)
    imN = permute(imN,[2 3 1]);
    for indG2 = imN
        for indz = 1:3
            NLoc = C(indG2,1,indz);
            if NLoc ~= 0 %execute if neighbor exists
                if GS(NLoc) == 0 %execute if neighbor unoccupied
                    NMaster(indG2) = NMaster(indG2)-1; %free up one neighbor
                end
            end
        end
    end
```

Figure 23. Code that checks for the neighbors of each existing atom, along with each of the neighbor's neighbors. These are later used to run the kMC.

```
%begin kMC atom movement
currentN = NMaster(indG); %find locations to move to
if currentN < 3 %execute only if the particle can move
    baselineEn = E(1, currentN+1); %determine energy to break bonds
    R = zeros(1,3);

    %generate rates for each event
    for indz = 1:3
        NLoc = NState(indz);
        if NLoc ~= 0 %execute only for possible positions
            neighborN = NMaster(NLoc); %retrieve number of neighbors for that neighbor
            En = baselineEn + E(2,neighborN); %add energy of diffusion/bond formation
            R(indz) =  v*exp(-En/(kbT)); %calculate rate of moving to that position
        end
    end

    %now select an event to execute

    %randomize the list, as it was previously sorted
    i = randperm(length(R));
    sortR = R(i);

    %create the cumulative sum
    cumR = cumsum(sortR);

    %select an event
    U1 = randn(); %draw random number
    Rpos = find(cumR<=U1*cumR(3),1,'last'); %get position at
    %which the condition is met

    if isempty(Rpos)==0 %catch cases where nothing happens

        %find which location the event corresponds to
        newState = find(R==sortR(Rpos));
        newStateU = randi(length(newState));
        newState = NState(newState(newStateU));

        %catch any potential errors (used for debugging; no final purpose)
        if newState > 0
            GS(indG) = 0; %remove current atom
            GS(newState) = 1; %move atom to new position
        end
    end
end
```

Figure 24. Code that runs the complete kMC movement algorithm, from determining event rates, to selecting an event to execute, to executing said event (as explained in Appendix B).

```
        end
    end

    %output the final CVD chamber
    im = zeros(102,102);
    im(map(GS==1)) = 1;
end
```

Figure 25. Code that ends open ifs/for loops and outputs the final result for the live script.

D Reprinted Figures (Select Plots Only)

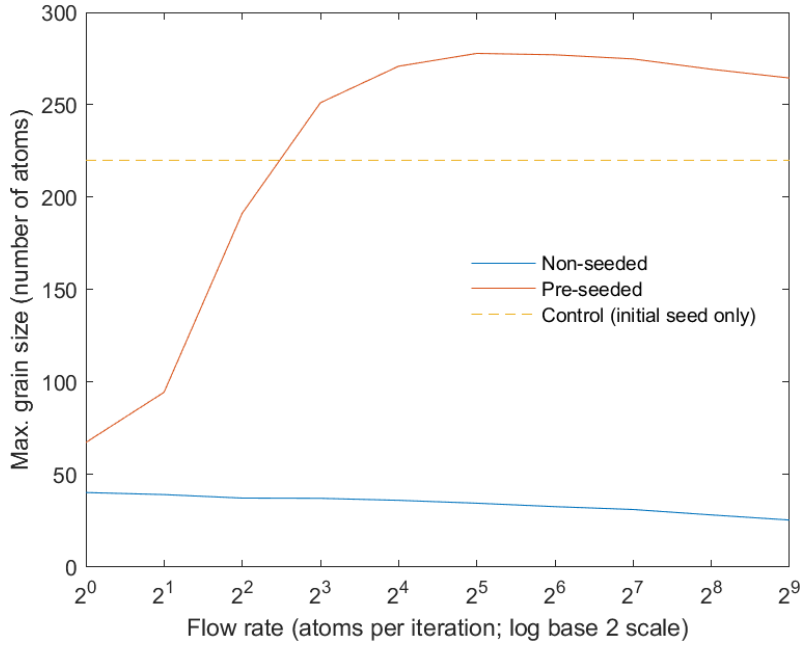

Figure 6. Mean maximum grain size by carbon flow rate, with a control (the atom count of the initial seed) (reprinted from page 4).

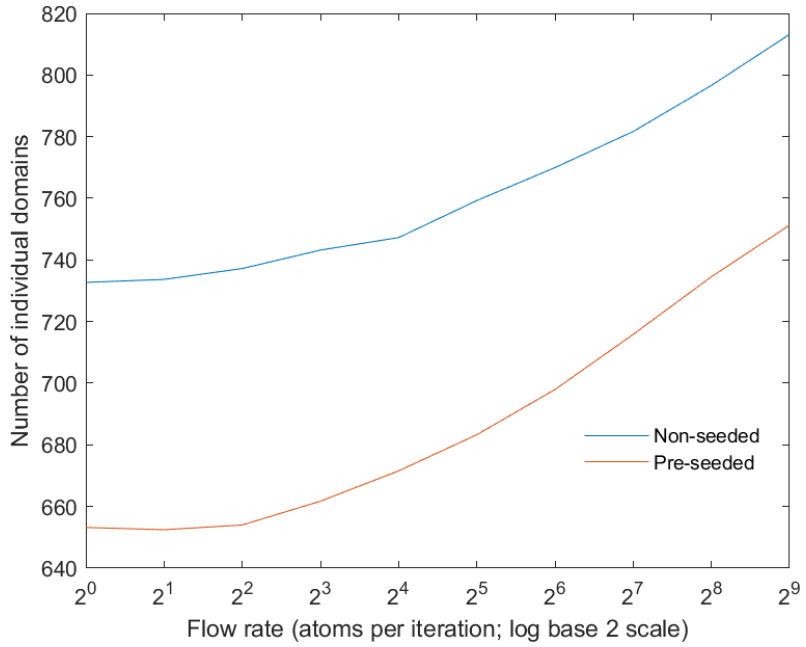

Figure 7. Mean grain count (number of individual grains) by carbon flow rate (reprinted from page 4).

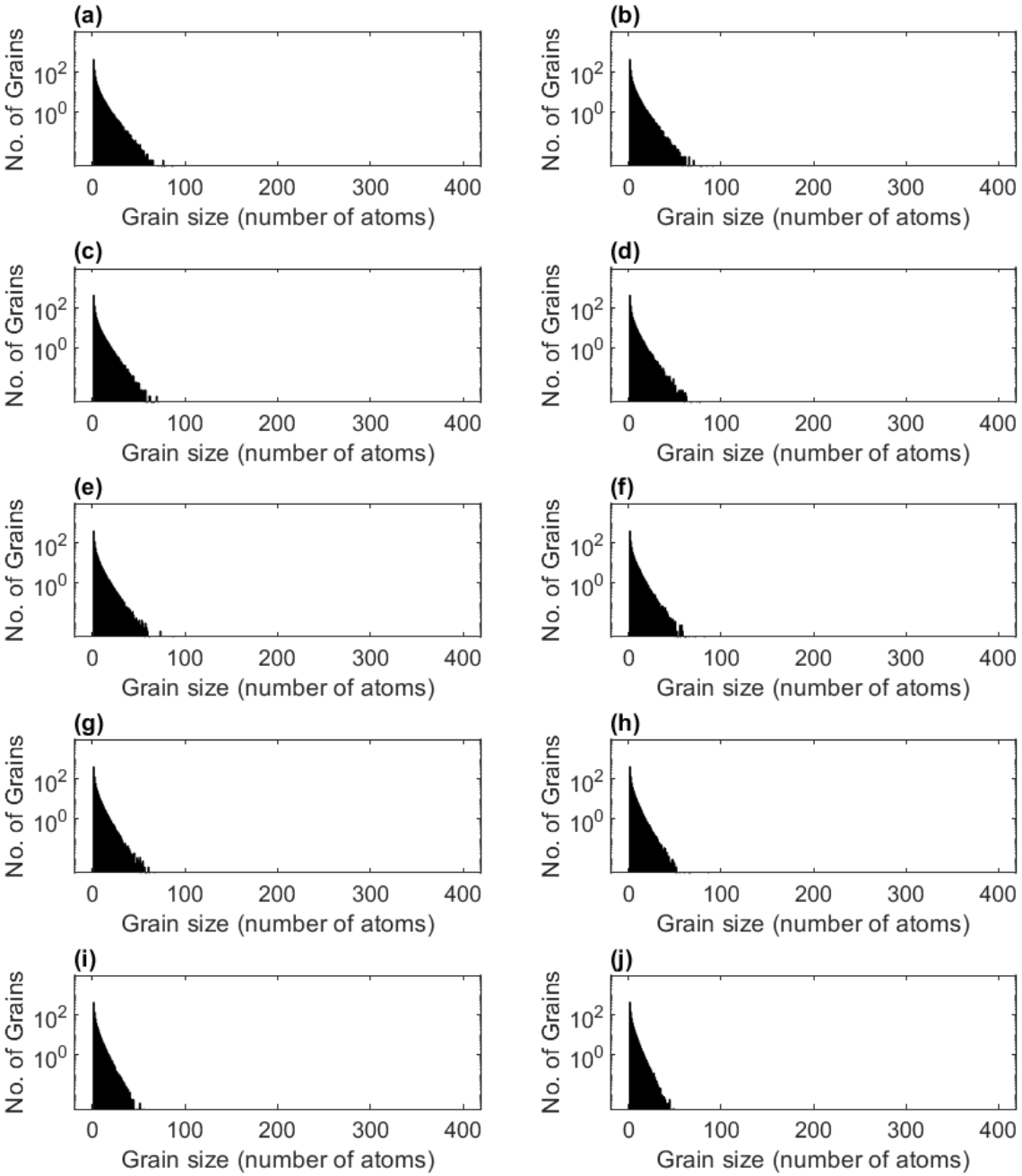

Figure 14. Histogram of mean non-seeded grain size distribution, for flow rates of (a) 2^0, (b) 2^1, (c) 2^2, (d) 2^3, (e) 2^4, (f) 2^5, (g) 2^6, (h) 2^7, (i) 2^8, (j) 2^9 *atoms per iteration (reprinted from page 6).*

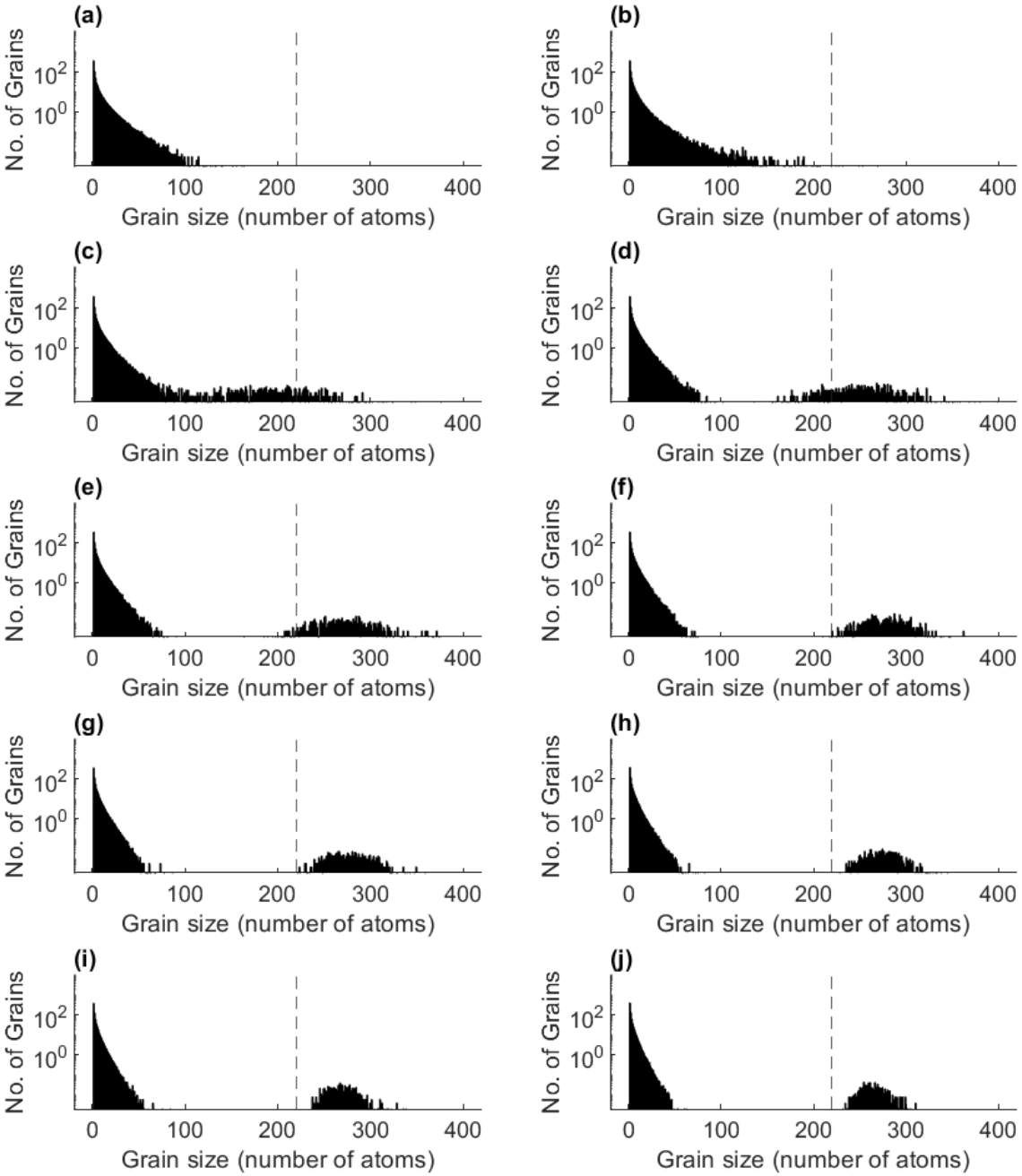

Figure 15. Histogram of mean pre-seeded grain size distribution, for flow rates of (a) 2^0, (b) 2^1, (c) 2^2, (d) 2^3, (e) 2^4, (f) 2^5, (g) 2^6, (h) 2^7, (i) 2^8, (j) 2^9 *atoms per iteration. Dashed lines indicate initial pre-seed size (reprinted from page 6).*